Cours Python ECG

I)Généralités

I.1)Variables et type

Les variables sont notées à l'aide d'une lettre ou d'un mot, suivi éventuellement d'un chiffre ou d'un caractère spécial.

Les principaux types de variables à connaître :

– le type **int** (=entier)
Exemple : a=2, temps=4, x1=−5

– le type **float** (=décimal)
Exemple : valeur_finale=3.5, resultat=10/3

– le type **bool** (=booléen)
Utilisé pour des variables dont la valeur est **True** ou **False**.

> Remarque
> Python est sensible à la casse, ce qui signifie qu'il fait la différence entre majuscule et miniscule.

I.2)Opérations

Entre des variables numériques a et b, on définit :

– l'addition a+b et la soustraction a-b

– la multiplication a*b et la division a/b

– la puissance a**n (où $n \in \mathbf{Z}$) [1]

– l'opération a//b, quotient de la division euclidienne de a par b,

– l'opération a%b, reste de la division euclidienne de a par b.

I.3)Fonction print

print(nom_de_variable_1,...,nom_de_variable_n)

Exemple

```
x=2
y=x+1
print(y)
print("la valeur de x est", x)
```

A l'écran, s'affiche 3 et << la valeur de x est 2 >>.

1. pour obtenir a^b dans le cas où $b \in \mathbf{R}$, on utilise l'instruction pow(a,b)

I.4)Fonction input

nom_de_variable=**input**(".....")

Exemple

```
mot=input("entrez un mot")
```

Python affiche le message << entrez un mot >> et enregistre la réponse de l'utilisateur sous la variable mot de type string (=chaîne de caractères).

Exemple

```
x=int(input("entrez un nombre entier"))
```

Python affiche le message << entrez un nombre entier >>, puis enregistre la réponse de l'utilisateur sous une variable de type string qu'il convertit en nombre entier par la commande **int**.

I.5)Incrémentation

L'incrémentation (respectivement la décrémentation) consiste à augmenter (respectivement diminuer) la valeur d'une variable numérique x d'une quantité donnée.

Exemple

```
x=5
x=x+1
```

A la fin du programme, x vaut 6.

Exemple

```
x=5
x=x-2
```

A la fin du programme, x vaut 3.

> Remarque
> Pour multiplier par 2 la valeur de x, c'est l'instruction : x=2*x.
> Pour diviser par 3 la valeur de x, c'est l'instruction : x=x/3.

II)Liste

Une liste est un tableau unidimensionnel qui contient une suite de valeurs de type entier, décimal, chaîne de caractères ou booléen.
Ces valeurs sont séparées par des virgules et placées entre crochets.

Exemple
```
animaux = ["girafe","tigre","gorille","souris"]
```
Exemple
```
taille=[500,100,175,5]
```
Exemple
```
mixte=["girafe",500,"tigre",100,"gorille",175,"souris",5].
```

II.1)Sélection d'un élément d'une liste

ma_liste[indice]

On se réfère à l'indice (ou index) de cet élément dans la liste avec la convention importante que **le premier élément de la liste porte l'indice zéro**.

Exemple
```
ma_liste=["âne","chèvre","fleur","wagon"]
```
Les commandes ma_liste[0],ma_liste[1], ma_liste[2] et ma_liste[3] renvoient respectivement les mots âne, chèvre, fleur et wagon.

Exemple
```
ma_liste=[5,10,7]
ma_liste[1]=3
print(ma_liste)
```
Ce programme remplace l'élément d'indice 1 de la liste par 3, puis affiche la liste modifiée : [5,3,7].

Remarque
On convient que le dernier élément d'une liste porte toujours l'indice -1.

II.2)Sélection d'éléments consécutifs d'une liste

ma_liste[i :j]

renvoie les éléments de la liste d'indices compris entre i et j-1.

Exemple
```
liste=[10,45,20,30,100,8,6]
print(liste[2:5])
```
La commande renvoie la liste : [20,30,100].

II.3)La commande append

ma_liste**.append**(valeur)

rajoute une valeur en fin de liste.

Exemple

```
ma_liste=[1,3,8,5]
ma_liste.append(12)
```

A l'issue, la liste vaut : [1,3,8,5,12].

II.4)La commande count

ma_liste**.count**(valeur)

compte le nombre d'occurences d'une valeur dans la liste.

Exemple

```
ma_liste=[13,4,13,2,13]
ma_liste.count(13)
```

A l'écran s'affiche 3 qui correspond au nombre d'occurences de l'élément 13 dans la liste.

II.5)La commande del

del ma_liste[indice]

supprime dans la liste l'élément d'indice donné.

Exemple

```
ma_liste=[5,4,8,15,12]
del ma_liste[2]
```

Cette commande supprime l'élément d'indice 2 de la liste, c'est-à-dire la valeur 8. A l'issue, la liste vaut donc [5,4,15,12].

II.6)La commande len

len(ma_liste)

compte le nombre d'éléments de la liste.

Exemple

```
maths=["théorème","constante","intégrale","moyenne"]
len(maths)
```

A l'écran s'affiche la valeur 4.

II.7)La commande in

Cette commande permet de construire une liste en compréhension.

Exemple

```
nombres=[k for k in range(5)]
```

La liste nombres est la liste : [0,1,2,3,4].

II.8)Concaténation de deux listes ou duplication

On peut concaténer deux listes en utilisant le symbole d'addition.

Exemple

```
a=[2,7,8]
b=[5,12]
print(a+b)
```

Le programme renvoie la liste : [2,7,8,5,12].

On peut aussi dupliquer une liste.

Exemple

```
liste=[0]
print(liste*4)
```

Le programme renvoie la liste : [0,0,0,0]

II.9)Maximum, minimum et somme

Dans une liste de nombres, on peut trouver son plus grand élément et son plus petit élément.
On peut également calculer la somme de ses éléments.

Exemple

```
a=[2,5,9,1]
print(max(a))
```

Le programme renvoie 9.

Exemple

```
a=[2,5,9,1]
print(min(a))
```

Le programme renvoie 1.

Exemple

```
a=[2,5,9,1]
print(sum(a))
```

Le programme renvoie 17.

III)Fonction range

Cette fonction retourne une suite d'entiers en progression arirhmétique.
Les paramètres d'entrée de la fonction range sont :
↪ start, premier entier de la suite,
↪ stop, l'entier jusqu'auquel on va, **sans l'inclure**,
↪ step, valeur entière positive ou négative qui représente l'écart algébrique entre deux entiers consécutifs de la suite, c'est-à-dire la raison.

> Remarque
> Pour transformer cette suite d'entiers en liste, on utilise la commande **list**, mais elle n'est pas au programme de la classe.

III.1)Fonction range à un paramètre

range(stop)

renvoie la suite des entiers consécutifs de 0 à stop-1.

Exemple

```
nombres=list(range(1,10))
print(nombres)
```

Ce programme retourne la liste [1,2,3,4,5,6,7,8,9].

III.2)Fonction range à deux paramètres

range(start,stop)

renvoie la suite des entiers consécutifs de start à stop- 1.

Exemple

```
nombres=list(range(-3,5))
print(nombres)
```

Ce programme retourne la liste [-3,-2,-1,0,1,2,3,4].

III.3)Fonction range à trois paramètres

range(start,stop,step)

renvoie les termes consécutifs de la suite arithmétique de premier terme start et de raison step, le dernier terme n'atteignant pas stop.

Exemple

```
nombres=list(range(5,20,3))
print(nombres)
```

Ce programme renvoie la liste [5,8,11,14,17].

Exemple

```
nombres=list(range(4,-5,-2))
print(nombres)
```

Ce programme renvoie la liste [4,2,0,-2,-4].

IV)Boucles

IV.1)Boucle for

for k **in** collection **:**
bloc d'instructions

exécute le bloc d'instructions autant de fois que de valeurs de k parcourant la collection (liste, objet de classe range, ...)

Exemple

```
for k in [1,2,3]:
    print(k)
```

A l'écran s'affichent 1 2 3 en colonne.

Exemple

```
s=0
for k in range(5):
    s=s+k
    print(s)
```

A l'écran s'affichent 0,1,3,6 et 10 qui sont les valeurs prises par s lors de l'exécution de la boucle.

IV.2)Boucle while

while condition **:**
bloc d'instructions

exécute le bloc d'instructions tant que la condition placée après while est réalisée.

Exemple

```
i=0
while i<100:
    print(i)
    i=i+1
```

A l'écran s'affichent en colonne tous les entiers consécutifs de 0 à 99.

> Remarque
> A la fin de la ligne for ou while, il faut **mettre deux points**, passer à la ligne, puis **indenter** de 4 espaces le bloc d'instructions de la boucle.

V)Instructions conditionnelles

Les instructions conditionnelles (ou tests) sont de trois types :
if ... ou **if...else...** ou **if...elif...else...**

La condition placée après **if**, **else** ou **elif** utilise les opérateurs suivants :

↪ **and** (et)

↪ **or** (ou)

↪ `==` (égal à)

↪ `<` (strictement inférieur à)

↪ `>` (strictement supérieur à)

↪ `! =` (différent de)

↪ `<=` (inférieur ou égal à)

↪ `>=` (supérieur ou égal à).

> Remarque
> Après la condition, il faut **mettre deux points**, passer à la ligne, puis **indenter** de 4 espaces les instructions du test.

V.1)Le test if...

if condition **:**
    bloc d'instruction(s)

exécute le bloc d'instructions si la condition est réalisée.

Exemple

```
age=int(input("entrez votre age"))
if age >= 18:
    print("vous êtes majeur")
```

V.2)Le test if...else...

if condition **:**
    bloc1 d'instruction(s)
else :
    bloc2 d'instruction(s)

exécute le bloc1 si la condition est réalisée ou le bloc2 si la condition n'est pas réalisée.

Exemple

```
age=int(input("entrez votre age"))
if age >= 18:
    print("vous êtes majeur")
else:
    print("vous êtes mineur")
```

V.3)Le test if...elif...else...

if condition1 **:**
    bloc1 d'instruction(s)
elif[2] condition2 **:**
    bloc2 d'instruction(s)
else :
    bloc3 d'instruction(s)

exécute le bloc1 et lui seul si la condition1 est réalisée, le bloc2 et lui seul si la condition1 n'est pas réalisée et la condition 2 est réalisée, le bloc3 et lui seul si les conditions 1 et 2 ne sont pas réalisées.
Dès qu'une condition est réalisée, on exécute le bloc et on sort du test.

Exemple

```
age=int(input("entrez votre age"))
if age >= 18:
    print("vous êtes majeur")
elif age <15:
    print("vous avez moins de 15 ans")
else:
    print("vous avez 15,16 ou 17 ans")
```

Remarque
Entre if et else, il est possible de mettre plusieurs elif.

VI)Fonctions

def nom_de_la_fonction(argument_1,...,argument_n) **:**
bloc d'instructions

Remarque
Pour obtenir la valeur renvoyée par la fonction, on utilise la commande **return**.

Exemple

```
def carre(x):
    return x**2
```

La commande carre(3) renvoie alors 3^2, c'est-à-dire 9.

Exemple

```
def carre(x):
    y=x**2
    return y
```

C'est le même programme, mais avec une variable locale y.

2. abréviation de else if

La présence d'arguments (= paramètres d'entrée) ou de paramètre de sortie (= ce qu'on retourne) dans une fonction n'est pas obligatoire.

Exemple

```
def compteur(stop):
    i= 0
    while i<stop:
        print(i)
        i=i+1
```

La commande compteur(4) affiche à l'écran 0,1,2,3.
Cette fonction ne possède pas de paramètre de sortie.

Exemple

```
def hello():
    print("bonjour")
```

La commande hello() affiche bonjour.
Cette fonction ne possède ni argument ni paramètre de sortie.

VII)Modules

Un module (ou bibliothèque) permet l'importation de fonctions dans des domaines variés (algèbre linéaire, analyse, probabilités, statistiques).

Les modules utilisés en ecg sont :
numpy, **numpy.linalg**, **numpy.random** et **matplotlib.pyplot**.

Pour accéder à un module ou à une fonction d'un module donné, on peut saisir au choix l'une des commandes suivantes :

– **import** nom_du_module **as** alias [3]

Toute fonction du module est alors utilisable par la commande :
alias.nom_de_la_fonction

– **from** nom_du_module **import** nom_de_la_fonction

Cela permet d'utiliser une fonction donnée du module.

– **from** nom_du_module **import** * [4]

Cela permet d'importer toutes les fonctions du module.

Exemple :

```
import numpy as np
A=np.array(([1,2],[3,4]))
print(A)
```

On a utilisé l'alias np pour désigner le module numpy, puis dans ce module, on a utilisé la fonction array.

Ce programme affiche la matrice $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$.

3. abréviation qu'on choisit pour désigner le module
4. En informatique, l'astérisque signifie « tout »

VII.1)Outils matriciels

Les fonctions ci-dessous proviennent du module **numpy** d'alias np.

np.**array**(([1, 2], [3, 4])) renvoie la matrice $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$,

np.**array**(([1], [3])) renvoie la matrice colonne $\begin{pmatrix} 1 \\ 3 \end{pmatrix}$,

np.**array**(([1, 2])) renvoie la matrice ligne $\begin{pmatrix} 1 & 2 \end{pmatrix}$,

np.**zeros**(**shape**=(2,3)) renvoie la matrice $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$,

np.**ones**(**shape**=(2,3)) renvoie la matrice $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$,

np.**eye**(2) renvoie la matrice $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$,

np.**transpose**(A) renvoie la transposée de la matrice A,

np.**shape**(A) renvoie le format de la matrice A,

np.**dot**(A,B) renvoie le produit de la matrice A par la matrice B.

> Remarque
> Pour la somme et la différence, on fait tout simplement $A + B$ et $A - B$. En revanche, l'opération $A * B$ renvoie non pas la matrice AB, mais la matrice de coefficient $c_{ij} = a_{ij}b_{ij}$.

VII.2)Elément, ligne ou colonne d'une matrice

Soit $A \in \mathscr{M}_{n,p}(\mathbf{R})$ une matrice.

A[i,j] renvoie le coefficient de la i-ème ligne et j-ième colonne de A.

A[i, :] renvoie la i-ème ligne de A.

A[:,j] renvoie la j-ème colonne de A.

> Remarque
> Les lignes et les colonnes sont numérotées à partir de l'indice zéro.

VII.3)Complément d'algèbre linéaire

Les fonctions ci-dessous viennent du module **numpy.linalg** d'alias al.

Soit A une matrice (carrée si nécessaire).

al.**inv**(A) renvoie la matrice inverse de A,

al.**matrix_rank**(A) renvoie le rang de A,

al.**matrix_power**(A,n) renvoie A^n,

al.**eig**(A) renvoie valeurs propres et les sous-espaces propres de A,

al.**solve**(A,B) renvoie la matrice X telle que AX=B où X et B sont des matrices colonnes.

VII.4)Statistiques et sommes

Les fonctions ci-dessous proviennent du module **numpy** d'alias np.

Si A est une liste ou une matrice :

np.**min**(A) renvoie le plus petit élément de la matrice A,

np..**max**(A) renvoie le plus grand élément de la matrice A,

np.**mean**(A) renvoie la moyenne des éléments de A,

np.**median**(A) renvoie la médiane de A,

np.**var**(A) renvoie la variance de A,

np.**std**(A) renvoie l'écart-type de A,

np.**sum**(A) renvoie la somme des coefficients de A,

np.**cumsum**(A) renvoie la liste des sommes partielles des valeurs de A.

Exemple :

```
import numpy as np
A=[1,5,8,11]
print(cumsum(A))
```

Le programme affiche la liste [1,6,14,25].

VII.5)Fonctions usuelles

Les fonctions ci-dessous proviennent du module **numpy** d'alias np.

np.**exp**(x) renvoie l'exponentielle de x,

np.**log**(x) renvoie le logarithme népérien de x,

np.**sqrt**(x) renvoie la racine carrée de x,

np.**abs**(x) renvoie la valeur absolue de x,

np.**floor**(x) renvoie la partie entière de x.

> Remarque
> Ces fonctions restent valables si x est une matrice ou une liste, la fonction s'appliquant alors élémént par élément.

VII.6)Nombres spéciaux

Nombres provenant du module **numpy** d'alias np :

np.**e** renvoie l'exponentielle de 1,

np.**pi** renvoie le nombre π.

VII.7)Probabilités

Les fonctions ci-dessous viennent du module **numpy.random** d'alias rd.

rd.**random()**, réel aléatoire entre 0 et 1,

rd.**binomial**(n,p), valeur d'une variable aléatoire suivant $\mathscr{B}(n,p)$,

rd.**randint**(a,b), valeur d'une variable aléatoire suivant $\mathscr{U}(\llbracket a, b-1 \rrbracket)$,
rd.**geometric**(p), valeur d'une variable aléatoire suivant $\mathscr{G}(p)$,
rd.**poisson**(λ), valeur d'une variable aléatoire suivant $\mathscr{P}(\lambda)$,
rd.**exponential**(λ), valeur d'une variable aléatoire suivant $\mathscr{E}(\lambda)$,
rd.**normal**(m,σ), valeur d'une variable aléatoire suivant $\mathscr{N}(m, \sigma^2)$,

VII.8)Vecteur à pas constant

Les deux fonctions ci-dessous proviennent du module **numpy** d'alias np. Elles renvoient un vecteur à pas constant, c'est-à-dire un vecteur ligne (ou matrice ligne) de type array dont les composantes sont des nombres en progression arithmétique.

np.**arange**(start,stop,step) où les paramètres d'entrée sont :
↪ start, premier nombre du tableau,
↪ stop, le nombre vers lequel on va, mais sans l'inclure,
↪ step, nombre positif ou négatif qui représente l'écart algébrique entre deux entiers consécutifs du tableau.

> Remarque
> La fonction arange est similaire à la fonction range à la différence que les paramètres d'entrée de la fonction arange sont de type float (=décimal), alors que ceux de la fonction range sont de type int (=entier).

np.**linspace**(start,stop,num) où les paramètres d'entrée sont :
↪ start, premier nombre du tableau,
↪ stop, dernier nombre du tableau
↪ num, le nombre de termes du tableau.

> Remarque
> Comme pour la fonction arange, les paramètres d'entrée de la fonction linspace sont de type float.

Exemple :
```
import numpy as np
a=np.arange(1,2.5,0.4)
print(a)
```
Le programme affiche la matrice ligne ligne $(1, 1.4, 1.8, 2.2)$.

Exemple :
```
import numpy as np
a=np.linspace(1,4,5)
print(a)
```
Le programme affiche la matrice ligne $(1, 1.75, 2.5, 3.25, 4)$.

VII.9)Représentations graphiques et statistiques

Les fonctions viennent du module **matplotlib.pyplot** d'alias plt.

plt.**plot**(x,y) trace la ligne brisée $(x_1, y_1) - (x_2, y_2) - \cdots - (x_n, y_n)$ où x et y sont des listes définies par $x = [x_1, x_2, ..., x_n]$ et $y = [y_1, y_2, ..., y_n]$.

plt.**bar**(x,y,e) trace un diagramme en bâtons où :
x est une liste formée de valeurs statistiques (qualitatives ou quantitatives) deux à deux distinctes,
y est une liste représentant l'effectif associé (hauteur du bâton),
e est l'épaisseur de chaque bâton.

plt.**hist**(x,n) ou plt.**hist**(x,y) trace un histogramme où :
x est une liste formée de valeurs statistiques quantitatives (pas forcément distinctes),
n est un entier égal au nombre de classes (de même amplitude),
y est une liste dont les éléments sont les extrémités des classes.

plt.**boxplot**(x) trace la boîte à moustache de la série statistique quantitative x.

plt.**show()** permet d'afficher les représentations graphiques à l'écran.